LAPORAN TUGAS BESAR II
IF2123 ALJABAR LINIER DAN GEOMETRI

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah
IF2123 Aljabar Linier dan Geometri

Disusun oleh:

Kelompok 21 - penCITRAan

13520004 Gede Prasidha Bhawarnawa

13520014 Muhammad Helmi Hibatullah

13520023 Ahmad Alfani Handoyo

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 1 TAHUN 2021/2022

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Kompresi gambar merupakan suatu tipe kompresi data yang dilakukan pada gambar digital. Dengan kompresi gambar, suatu file gambar digital dapat dikurangi ukuran filenya dengan baik tanpa mempengaruhi kualitas gambar secara signifikan. Terdapat berbagai metode dan algoritma yang digunakan untuk kompresi gambar pada zaman modern ini. Salah satu algoritma yang dapat digunakan untuk kompresi gambar adalah algoritma SVD (Singular Value Decomposition).

Buatlah program kompresi gambar dengan memanfaatkan algoritma SVD dalam bentuk website lokal sederhana. Spesifikasi website adalah sebagai berikut:

1. Website mampu menerima file gambar beserta input tingkat kompresi gambar (dibebaskan formatnya).
2. Website mampu menampilkan gambar input, output, runtime algoritma, dan persentase hasil kompresi gambar (perubahan jumlah pixel gambar).
3. File output hasil kompresi dapat diunduh melalui website.
4. Kompresi gambar tetap mempertahankan warna dari gambar asli.
5. (Bonus) Kompresi gambar tetap mempertahankan transparansi dari gambar asli, misal untuk gambar png dengan background transparan.
6. Bahasa pemrograman yang boleh digunakan adalah Python, Javascript, dan Go.
7. Penggunaan framework untuk back end dan front end website dibebaskan. Contoh framework website yang bisa dipakai adalah Flask, Django, React, Vue, dan Svelte.
8. Kalian dapat menambahkan fitur fungsional lain yang menunjang program yang anda buat (unsur kreativitas diperbolehkan/dianjurkan).
9. Program harus modular dan mengandung komentar yang jelas.
10. Diperbolehkan menggunakan library pengolahan citra seperti OpenCV2, PIL, atau image dari Go.
11. Dilarang menggunakan library perhitungan SVD dan library pengolahan eigen yang sudah jadi.

BAB II
TEORI SINGKAT

Perkalian Matriks

Perkalian matriks terbagi menjadi dua, yaitu perkalian matriks dengan matriks dan perkalian matriks dengan skalar. Pada perkalian dua buah matriks, misalkan A adalah sebuah matriks m x r dengan elemennya adalah a_{ij} dan B adalah sebuah matriks r x n dengan elemennya b_{ij}, maka hasil perkalian A x B adalah sebuah matriks C berukuran m x n yang tiap elemennya, c_{ij}, dapat dihitung menggunakan rumus berikut.

$$c_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + \ldots + a_{in}.b_{nj}$$

Pada perkalian matriks dengan skalar, misalkan A adalah sebuah matriks dan c adalah sebuah skalar, maka hasil perkalian cA adalah sebuah matriks yang tiap elemen di A dikali dengan skalar c.

Nilai Eigen dan Vektor Eigen

Jika A adalah matriks n x n maka vektor tidak-nol di R^n disebut vektor eigen dari A jika Ax sama dengan perkalian suatu skalar λ dengan x, yaitu Ax = λx. Skalar λ disebut nilai eigen dari A, dan x dinamakan vektor eigen yang berkorespondensi dengan λ. Nilai eigen menyatakan nilai karakteristik dari sebuah matriks yang berukuran n x n. Vektor eigen x menyatakan vektor kolom yang apabila dikalikan dengan sebuah matriks n x n menghasilkan vektor lain yang merupakan kelipatan vektor itu sendiri.

Matriks SVD

Algoritma SVD didasarkan pada teorema dalam aljabar linier yang menyatakan bahwa sebuah matriks dua dimensi dapat dipecah menjadi hasil perkalian dari 3 sub-matriks yaitu matriks ortogonal U, matriks diagonal S, dan transpose dari matriks ortogonal V. Dekomposisi matriks ini dapat dinyatakan sesuai persamaan berikut.

$$A_{m \times n} = U_{m \times m} S_{m \times n} V^T_{n \times n}$$

Matriks U adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks AA^T . Matriks ini menyimpan informasi yang penting terkait baris-baris matriks awal, dengan informasi terpenting disimpan di dalam kolom pertama. Matriks S adalah matriks

diagonal yang berisi akar dari nilai eigen matriks U atau V yang terurut menurun. Matriks V adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks A^TA. Matriks ini menyimpan informasi yang penting terkait kolom-kolom matriks awal, dengan informasi terpenting disimpan dalam baris pertama.

Gambar 1. Ilustrasi Algoritma SVD dengan rank k

Dapat dilihat di gambar di atas bahwa dapat direkonstruksi gambar dengan banyak singular values k dengan mengambil kolom dan baris sebanyak k dari U dan V serta singular value sebanyak k dari S atau Σ terurut dari yang terbesar. Kita dapat mengaproksimasi suatu gambar yang mirip dengan gambar aslinya dengan mengambil k yang jauh lebih kecil dari jumlah total singular value karena kebanyakan informasi disimpan di singular values awal karena singular values terurut mengecil. Nilai k juga berkaitan dengan rank matriks karena banyaknya singular value yang diambil dalam matriks S adalah rank dari matriks hasil, jadi dalam kata lain k juga merupakan rank dari matriks hasil. Maka itu matriks hasil rekonstruksi dari SVD akan berupa informasi dari gambar yang terkompresi dengan ukuran yang lebih kecil dibanding gambar awal.

BAB III
IMPLEMENTASI

Program Sisi Peladen

Sisi peladen atau *server side* yang ditangani pada *app.py* mempunyai fungsi untuk menangani segala permintaan dari pengguna dan menghubungkannya dengan program algoritma kompresi pada backend. Di sini digunakan *framework* Flask (terdapat pada file app.py) karena kepraktisannya dalam menangani web yang sederhana dan juga kemampuannya untuk menjalankan program Python yang digunakan untuk menjalankan algoritma kompresi.

1. Inisialisasi library

```
from flask import Flask, flash, request, redirect, url_for, render_template, Markup
import os
from werkzeug.utils import secure_filename
from os.path import join, dirname, realpath
from compressor import compressor
```

Pada baris pertama diambil dari library flask fungsi dan objek Flask, flash, request, redirect, url_for, render_template, dan Markup. Objek *Flask* digunakan untuk membuat *instance* dari dirinya untuk membuat aplikasi WSGI. *flash* digunakan untuk menunjukkan suatu pesan di *request* selanjutnya. *request* digunakan untuk mengakses data permintaan yang masuk dari klien. *redirect* digunakan untuk mengarahkan klien ke suatu halaman tujuan. *url_for* digunakan untuk membuat URL untuk argumen yang diberikan kepadanya. *render_template* menampilkan halaman HTML yang telah dibuat pada folder *templates*. *Markup* digunakan untuk memastikan string di dalamnya sesuai untuk dimasukkan pada HTML.

Pada baris kedua dan keempat diambil library os yang digunakan untuk mengakses directory dan komunikasi dengan OS. *join*, *dirname,* dan *realpath* secara kolektif akan digunakan untuk menghasilkan alamat suatu file relatif ke letak di mana program *app.py* dijalankan

Pada baris ketiga diambil *secure_filename* dari *werkzeug.utils* yang digunakan sebagai rekomendasi dari dokumentasi Flask untuk mengakses nama file yang telah diunggah pada website.

Pada baris terakhir diimpor fungsi compressor dari /compressor.py yang memegang algoritma proses kompresi gambar.

2. Inisialisasi *instance* Flask

```
app = Flask(__name__, static_folder=join(dirname(realpath(__file__)), 'static/'))
```

Pada baris ini dibuatlah *instance* Flask yang akan menangani segala permintaan dari klien dan menghubungkannya dengan backend. Diberi argumen *static_folder* untuk memperjelas dimanakah letak dari folder *static* yang memegang gambar-gambar, elemen Bootstrap, dan elemen-elemen lainnya yang dibutuhkan website.

3. Inisialisasi folder gambar original dan terkompres

```
# MEMBUAT DIR KALAU BELUM ADA
ORIGINAL_FOLDER = join(dirname(realpath(__file__)), 'static/original/')
os.makedirs(ORIGINAL_FOLDER, exist_ok=True)
COMPRESSED_FOLDER = join(dirname(realpath(__file__)), 'static/compressed/')
os.makedirs(COMPRESSED_FOLDER, exist_ok=True)
```

Bila folder *static/original/* dan *static/compressed/* belum ada pada directory, maka akan dibuat terlebih dahulu. Folder-folder inilah yang akan menyimpan gambar-gambar yang yang akan diunggah dan diunduh oleh klien nantinya.
Dibuat juga variabel yang menandakan directory folder gambar original dan yang terkompress pada *ORIGINAL_FOLDER* dan *COMPRESSED_FOLDER*.

4. Mengatur variabel dan fungsi tambahan

```
app.secret_key = "pencitraan"
app.config['ORIGINAL_FOLDER'] = ORIGINAL_FOLDER
app.config['COMPRESSED_FOLDER'] = COMPRESSED_FOLDER
app.config['PREFIX_COMP'] = "compressed_"
app.config['MAX_CONTENT_LENGTH'] = 120 * 1024 * 1024

ALLOWED_EXTENSIONS = set(['bmp', 'dds', 'dib', 'eps', 'gif', 'icns', 'ico',
'im', 'jpeg', 'jpg', 'msp', 'pcx', 'png', 'pbm', 'sgi', 'spi', 'tga',
'tiff', 'webp', 'xbm'])
```

secret_key digunakan untuk mengakses *cookie* yang dibutuhkan website pada satu sesi tertentu. Nilainya dapat diatur menjadi apa pun. Diatur konfigurasi untuk *instance* Flask app untuk *ORIGINAL_FOLDER* dan *COMPRESSED_FOLDER* agar nantinya dapat dipakai pada route-route nantinya. Diatur juga konfigurasi prefiks yang akan ditambahkan ke nama file yang telah terkompres dan juga besar permintaan maksimum yang dapat diterima, yaitu 120*1024*1024 bytes atau sekitar 125 MB. *ALLOWED_EXTENSIONS* menyimpan segala tipe file yang dapat diterima program berdasarkan tipe file yang didukung oleh library PILLOW dalam pengolahan gambar.

```
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Fungsi *allowed_file* mengecek apakah nama file yang diterima mengandung tipe file yang didukung.

5. Route dasar

```
@app.route("/")
def home():
    return render_template("home.html")
```

```
@app.route('/how-to-use')
def howtouse():
    return render_template("howtouse.html")

@app.route('/about-us')
def aboutus():
    return render_template("aboutus.html")
```

Route-route dasar ini bagaikan alamat-alamat web yang bila diakses melalui browser, akan menampilkan HTML untuk masing-masing render_templatenya.
Bila web lokal, maka alamat indeks kemungkinan besar akan berada di http://127.0.0.1:5000/ Sehingga, ketika kita mengakses http://127.0.0.1:5000/ maka akan ditampilkan *home.html*. Bila mengakses http://127.0.0.1:5000/how-to-use akan ditampilkan *howtouse.html*, dan http://127.0.0.1:5000/about-us menampilkan *aboutus.html*.

6. Route yang menangani request unggah gambar

```python
@app.route('/', methods=['POST'])
def process_image():
    # GET REQUEST FILE DAN COMPRESSION RATE
    file = request.files['file']
    comprate = request.form.get('comp-rate', type=int)
    if allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['ORIGINAL_FOLDER'], filename))

        # FUNGSI COMPRESSOR MENGEMBALIKAN DUA NILAI: "return (runtime, comprate)"
        runtime, comprate = compressor(app.config['ORIGINAL_FOLDER'], app.config['COMPRESSED_FOLDER'], filename, comprate, app.config['PREFIX_COMP'])
        runtime = round(runtime, 4)
        comprate = round(comprate, 2)

        return render_template('home.html', filename=filename, runtime=runtime, cprate=comprate)
    else:
        flash(Markup('Allowed image types are as specificied <a href="http://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html" target="_blank">here.</a>'))
        return redirect(request.url)
```

Route ini digunakan untuk menerima request bermetode POST yang mengandung data formulir dari klien dan menjalankan program kompresi gambar.

Pertama didapatkan dulu data formulir untuk file dan tingkat kompresi menggunakan request.files untuk input bernama ‘file’ dan request.form.get untuk input bernama comp-rate yang *di-casting* menjadi *integer*.Dicek menggunakan fungsi *allowed_file* apakah nama file diperbolehkan.

Bila tidak, maka akan ditunjukkan pesan flash dengan string yang dicek dengan Markup untuk memberikan peringatan ke klien. Klien kemudian diarahkan kembali ke URL yang membuat request tersebut.

Bila iya diperbolehkan, maka pertama didapatkan nama file yang aman (menggunakan *secure_filename* dari *werkzeug.utils*) dan file akan disimpan pada directory */static/original/*. Algoritma kompresi gambar kemudian dijalankan pada *compressor* dengan argumen yang menyatakan letak folder original, folder hasil kompresi, nama file, tingkat kompresi, dan prefiks untuk nama file gambar terkompresi. Algoritma *compressor* akan mengembalikan tuple nilai *runtime* dan persentase kompresi yang dibulatkan. Terakhir, akan ditampilkan *home.html* dengan argumen filename, runtime, dan cprate.

7. Route yang menangani display gambar

```python
@app.route('/original/<filename>')
def display_org(filename):
    return redirect(url_for('static', filename='original/' + filename), code=301)

@app.route('/compressed/<filename>')
def display_comp(filename):
    return redirect(url_for('static', filename='compressed/' + app.config['PREFIX_COMP']+ filename), code=301)
```

Ada dua route tambahan yang digunakan untuk membantu mendapatkan *URL* untuk gambar original dan terkompresi untuk ditampilkan ke klien. Didapatkan URL untuk alamat gambar original dan terkompresi dengan *url_for* dengan kode 301. Route ini berguna juga untuk tombol 'Download' karena membutuhkan adanya link dari gambar dan route menyediakan link tersebut.

8. Menjalankan Flask

```
if __name__ == "__main__":
    app.run(debug=True)
```

Flask dapat dijalankan bila kita menjalankan app.py secara langsung, *app.run* akan memulai *instance* Flask. Argumen debug=True berarti bila kita mengubah isi dari file .py yang mengandung Flask tersebut, maka Flask akan langsung *restart*.

Program Utama

Program utama terbagi menjadi dua, yaitu compressor.py dan svd_qr.py. Program compressor.py merupakan program berisi sebuah fungsi bernama compressor untuk memproses gambar. Sedangkan program svd_qr.py hanya berisi fungsi-fungsi untuk mengimplementasikan algoritma SVD.

1. compressor.py
    a. Import library

```
1  from PIL import Image
2  from os.path import join
3
4  import numpy as np
5  import time
6  import svd_qr
```

Pada bagian ini compressor.py mengimport library dan fungsi yang akan dipakai dalam program. Library yang dipakai antara lain PIL, os, numpy, time, dan svd_qr. Pertama, PIL digunakan untuk membuka gambar masukan dan menyimpan gambar keluaran serta mengubah mode gambar jika diperlukan. Kemudian, library os diimport agar dapat menggunakan fungsi *join* untuk menyatukan *path* serta nama file yang ada pada program sisi peladen. Selanjutnya, library numpy digunakan untuk mengubah gambar masukan menjadi bentuk matriks. Library time digunakan untuk menghitung *runtime*. Terakhir, svd_qr merupakan program buatan sendiri untuk mengkompres gambar dengan algoritma SVD.

b. Fungsi compressor

```
8   def compressor(original_realpath, compressed_realpath, file_name, cprate, prefix):
9       start = time.time()
10
11      img_in = Image.open(join(original_realpath, file_name))
12
13      # Cek mode gambar, jika L, P, atau PA, ubah ke RGB atau RGBA
14      im_mode = img_in.mode
15      if im_mode == 'L':
16          img_in = img_in.convert('RGB')
17      elif im_mode == 'LA':
18          img_in = img_in.convert('RGBA')
19      elif im_mode == 'P':
20          img_in = img_in.convert('RGB')
21      elif im_mode == 'PA':
22          img_in = img_in.convert('RGBA')
23
24      # Ubah gambar menjadi array
25      img_array = np.array(img_in)
26
27      # Cek length dan width dari gambar untuk menghitung k
28      length, width, _ = img_array.shape
29      max_rank = min(length, width)
30      k = (cprate * max_rank) // 100
31
32      # Hitung kompresi gambar dengan SVD
33      img_compressed = svd_qr.matriximage(img_array, k)
34
35      # Buat kembali gambar dari matriks
36      img_out = Image.fromarray(img_compressed, mode=img_in.mode)
37
38      # Jika mode gambar awalnya adalah L, P, atau PA, ubah kembali
39      # menjadi L, P, atau PA
40      if im_mode == 'L':
41          img_out = img_out.convert('L')
42      elif im_mode == 'LA':
43          img_out = img_out.convert('LA')
44      elif im_mode == 'P':
45          img_out = img_out.convert('P')
46      elif im_mode == 'PA':
47          img_out = img_out.convert('PA')
48
```

```
    # Save
    img_out.save(join(compressed_realpath, prefix + file_name))

    # Hitung runtime
    end = time.time()
    runtime = end - start

    # Hitung pixel difference
    pixel_diff = (length * k + k + width * k) / (length * width) * 100

    img_in.close()

    return runtime, pixel_diff
```

Fungsi compressor menerima parameter berupa original_realpath, compressed_realpath, file_name, cprate, dan prefix. Fungsi ini akan mengembalikan waktu *runtime* dan *pixel difference* untuk digunakan pada program sisi klien.

Algoritma dari fungsi ini dimulai dengan menetapkan waktu dimulainya *runtime*. Kemudian membuka berkas gambar dengan *path* yang didapat dari program sisi peladen dan mengubah mode gambar menjadi RGB atau RGBA jika mode gambar sebelumnya adalah L (*luminance*), LA (*luminance* dengan alpha), P (*palette*) atau PA (*palette* dengan alpha). Setelah itu, gambar diubah menjadi ke dalam bentuk matriks. Kemudian dicari panjang dan lebar dari matriks tersebut untuk dicari nilai k yang sesuai untuk dilakukan pemotongan. Selanjutnya, matriks dihitung nilai SVD nya dengan nilai k yang didapat sebelumnya dengan memanggil fungsi matriximage dari program svd_qr. Setelah itu, gambar dibuat kembali dari matriks. Kemudian, mode gambar diubah lagi jika perlu dan kemudian disimpan pada *path* yang didapat dari program sisi peladen. Terakhir, dihitung *runtime* keseluruhan algoritma dan juga dihitung *pixel difference* dari gambar yang yang telah dikompres. Pixel difference adalah suatu perbandingan antara jumlah pixel setelah diproses dengan jumlah pixel sebelum gambar diproses. Jumlah pixel dapat berubah meskipun gambar memiliki resolusi yang sama karena jumlah nilai singular yang diproses bisa saja berbeda. Perbandingan itu dinyatakan dengan rumus pixel_diff dengan nilai k sebagai rank (pemotong matriks singular).

2. svd_qr.py

a. Fungsi svd

```python
def svd(matrix):
    # BIKIN AT*A
    A = matrix.transpose() @ matrix

    # QR ALGORITHM AT*A untuk konvergen mendapatkan eigenvalue A dan eigenvector V
    for i in range(1):
        # DIGUNAKAN BUILT-IN FUNGSI QR DECOMPOSITION, asumsi boleh karena tidak langsung menghasilkan SVD
        Q, R = np.linalg.qr(A)
        A = R @ Q
        if not i:   # iterasi pertama inisialisasi V dengan Q
            V = Q
        else:
            V = V @ Q   # iterasi selanjutnya V = Q1 * Q2 ... * QN

    # DAPATKAN SIGMA
    sigma = np.diag(np.clip(np.sqrt(np.absolute(np.diagonal(A))), 1e-4,np.Inf))

    # Hitung inv(sigma) untuk menemukan U
    invsigma = np.linalg.inv(sigma)

    # U dari U = A * V * inv(S),  dapat asumsi V^T^-1 = V karena V orthogonal
    U = (matrix @ V) @ invsigma

    # TRANSPOSE V
    V = np.transpose(V)

    return U, sigma, V
```

Fungsi svd hanya menerima satu parameter, yaitu matrix. Parameter matrix merupakan matriks yang akan dicari matriks SVD-nya. Fungsi ini mengembalikan tuple berisi tiga matriks, yaitu matriks U, matriks sigma (S), dan matriks V^T.

Kami mengimplementasikan algoritma *QR Algorithm* yang pada prinsipnya adalah algoritma untuk mencari aproksimasi nilai eigen dan vektor eigen dengan melakukan *QR Decomposition* pada setiap iterasinya. Algoritma pada fungsi ini dimulai dengan mentranspose matriks kemudian dikalikan dengan dirinya sendiri untuk membentuk matriks A^TA. Setelah itu, dilakukan pengulangan untuk mencari aproksimasi nilai eigen (yang ditampung pada variabel A) dan vektor eigen (yang ditampung pada variabel V).

Pada perulangan pertama, matriks A akan dilakukan dekomposisi QR dengan menggunakan fungsi yang sudah ada pada library numpy. Kemudian matriks A akan diubah nilainya dengan hasil perkalian dari matriks R dan Q. Elemen-elemen pada diagonal utama matriks A merupakan aproksimasi dari nilai-nilai eigen yang ada pada matriks A sebelum dilakukan dekomposisi. Lalu untuk matriks V akan diisi dengan nilai dari matriks Q yang merupakan aproksimasi dari vektor eigen. Untuk perulangan

selanjutnya, lakukan hal yang sama. Namun untuk matriks V akan diisi dengan perkalian dirinya sendiri dengan matriks Q.

Setelah dilakukan pengulangan, kita ambil nilai-nilai pada diagonal utama matriks A, cari masing-masing nilai mutlaknya, akarkan masing-masing nilainya, kemudian bentuk matriks diagonal dari nilai-nilai tersebut untuk mendapatkan matriks sigma. Di sini karena matriks sigma setelahnya akan diinvers, maka matriks sigma perlu dijadikan matriks persegi dulu. Juga, karena ada kemungkinan diagonal ada yang 0 sehingga determinan matriks menjadi 0 dan membuat matriks sigma tidak mempunyai invers, maka nilai diagonalnya perlu di-*clip* sehingga nilai minimalnya tidak 0 (1e-4 atau 10^{-4}). Untuk mendapatkan matriks U, kita lakukan perkalian matriks awal, matriks V, dan invers dari matriks S secara berurutan. Terakhir kita mentranspose matriks V untuk membentuk matriks V^T.

Kami memutuskan untuk menggunakan fungsi yang sudah ada untuk melakukan dekomposisi QR karena pada saat mencoba mengimplementasikannya sendiri dengan *Householder Reduction*, akan ada penurunan yang signifikan pada efisiensinya. Selain itu, penggunaan fungsi yang sudah ada untuk menghitung QR tidak menyalahi aturan pada spesifikasi tugas karena tidak langsung menghasilkan matriks SVD. Algoritma yang kami susun hanya melakukan satu kali pengulangan *QR Algorithm* pada pencarian nilai eigen dan vektor eigen karena aproksimasi yang dihasilkan sudah mendekati nilai yang sebenarnya. Selain itu, efisiensi dari algoritma ini juga meningkat karena proses terlama dari algoritma ini, yaitu dekomposisi QR dan perkalian matriks untuk menghasilkan variabel A dan V, hanya dilakukan sekali untuk matriks A^TA. Hal ini akan dijelaskan lebih lanjut pada bagian eksperimen.

b. Fungsi matriximage

```python
# TO PROCESSS THE DIFFERENT COLOR CHANNELS
# SUPPORT L, LA, RGB, RGBA, CMYK
# P and PA needs to be converted to RGB first
def matriximage(colormatrix, rank):
    m = colormatrix.shape[0]
    n = colormatrix.shape[1]
    col_channels = colormatrix.shape[2]

    img0 = np.zeros([m,n,col_channels])

    if col_channels > 0:
        c1 = np.array([[colormatrix[i][j][0] for j in range(n)] for i in range(m)]).astype(int)
        c11, c12, c13 = svd(c1)
        c11_ranked = c11[:, 0:rank]
        c12_ranked = c12[0:rank,0:rank]
        c13_ranked = c13[0:rank, :]
        c1_f = np.clip(((c11_ranked @ c12_ranked) @ c13_ranked).round(0), 0, 255)
        img0[:,:,0] = c1_f
    if col_channels > 1:
        c2 = np.array([[colormatrix[i][j][1] for j in range(n)] for i in range(m)]).astype(int)
        c21, c22, c23 = svd(c2)
        c21_ranked = c21[:, 0:rank]
        c22_ranked = c22[0:rank,0:rank]
        c23_ranked = c23[0:rank, :]
        c2_f = np.clip(((c21_ranked @ c22_ranked) @ c23_ranked).round(0), 0, 255)
        img0[:,:,1] = c2_f
    if col_channels > 2:
        c3 = np.array([[colormatrix[i][j][2] for j in range(n)] for i in range(m)]).astype(int)
        c31, c32, c33 = svd(c3)
        c31_ranked = c31[:, 0:rank]
        c32_ranked = c32[0:rank,0:rank]
        c33_ranked = c33[0:rank, :]
        c3_f = np.clip(((c31_ranked @ c32_ranked) @ c33_ranked).round(0), 0, 255)
        img0[:,:,2] = c3_f
    if col_channels > 3:
        c4 = np.array([[colormatrix[i][j][3] for j in range(n)] for i in range(m)]).astype(int)
        c41, c42, c43 = svd(c4)
        c41_ranked = c41[:, 0:rank]
        c42_ranked = c42[0:rank,0:rank]
        c43_ranked = c43[0:rank, :]
        c4_f = np.clip(((c41_ranked @ c42_ranked) @ c43_ranked).round(0), 0, 255)
        img0[:,:,3] = c4_f

    return np.uint8(img0)
```

Fungsi matriximage menerima dua parameter, yaitu colormatrix dan rank. Parameter colormatrix merupakan matriks yang akan dicari kompresinya dan rank merupakan ukuran seberapa banyak matriks dipotong. Fungsi ini akan mengembalikan matriks yang sudah dikompres dengan tipe data pada tiap elemennya berupa uint8.

Algoritma pada fungsi ini dimulai dengan menyimpan dimensi matriks serta jumlah *channel* pada matriks gambar. Jika gambar yang dikompres merupakan gambar dengan mode hitam putih, maka *channel*-nya hanya satu, untuk gambar dengan mode RGB, maka *channel*-nya ada tiga, dan seterusnya untuk setiap mode gambar. Kemudian, dilakukan inisialisasi matriks nol untuk menampung hasil

perhitungan. Pada setiap blok kode, akan dilakukan pemisahan matriks berdasarkan *channel*-nya kemudian dilakukan *casting* tipe data elemen matriks dari uint8 ke int. Setelah dipisah, akan dicari ketiga matriks SVD dari matriks yang sudah dipisah tadi dengan menggunakan fungsi svd. Setelah itu, ketiga matriks akan dipotong berdasarkan parameter rank. Kemudian, matriks akan disatukan kembali, dibulatkan menjadi nilai bilangan bulat, lalu di-*clip* agar nilainya berada dalam rentang 0-255. Terakhir, matriks dari masing-masing blok akan disatukan kembali pada matriks yang sudah diinisialisasi sebelumnya dan diubah kembali tipe data tiap elemennya dari int ke uint8.

Program Sisi Klien

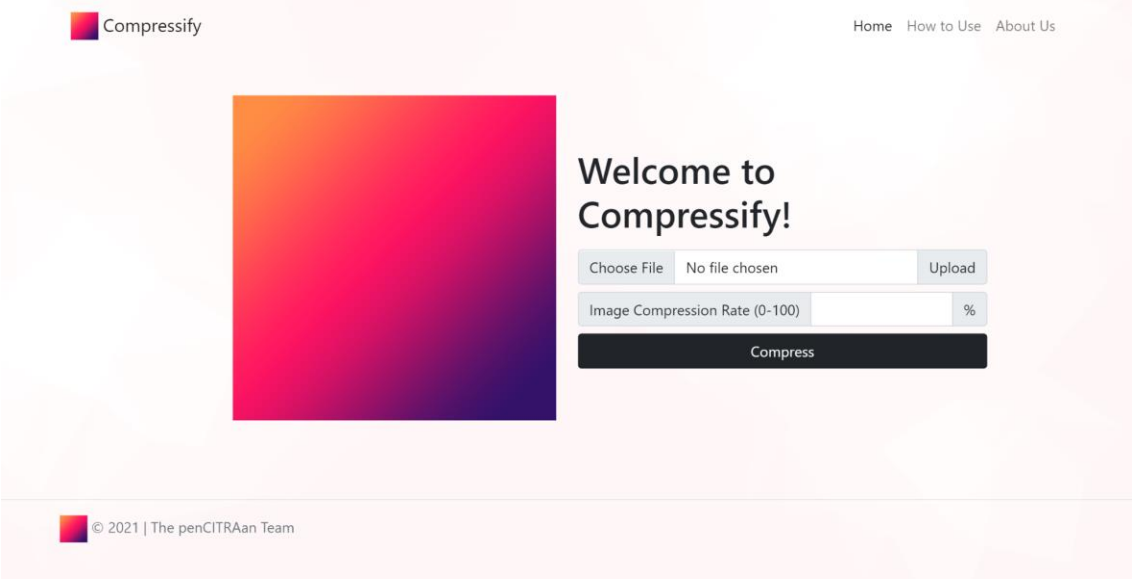
Sisi klien atau *client-side* adalah sisi yang menampilkan aplikasi web yang mengandung program utama untuk ditampilkan ke pengguna. Ini termasuk UI dari aplikasi web dan juga aksi-aksi yang dapat dilakukan oleh pengguna untuk berinteraksi dengan program. Di sini digunakan *framework CSS* Bootstrap sehingga mendukung pembuatan web secara cepat dengan tampilan yang elegan.

Di setiap halaman pada website terdapat bar navigasi di ujung atas yang mengalihkan user antara halaman Home, How to Use, dan About Us. Terdapat juga footer di bawah yang mengandung cap “The penCITRAan Team” yang bila diklik mengalihkan user kembali ke Home.

Keterangan: pada beberapa screenshot dibawah terlihat bahwa beberapa background tidak mencakupi seluruh halaman yang pada nyatanya seharusnya mencakup seluruh halaman. Ini dikarenakan gambar background diatur sehingga background tidak bergerak bila halaman di-*scroll*. Ketidakpenuhan background ini berupa sebuah efek samping menggunakan fitur *“Capture full size screenshot”* pada *Developer Mode* di Google Chrome.

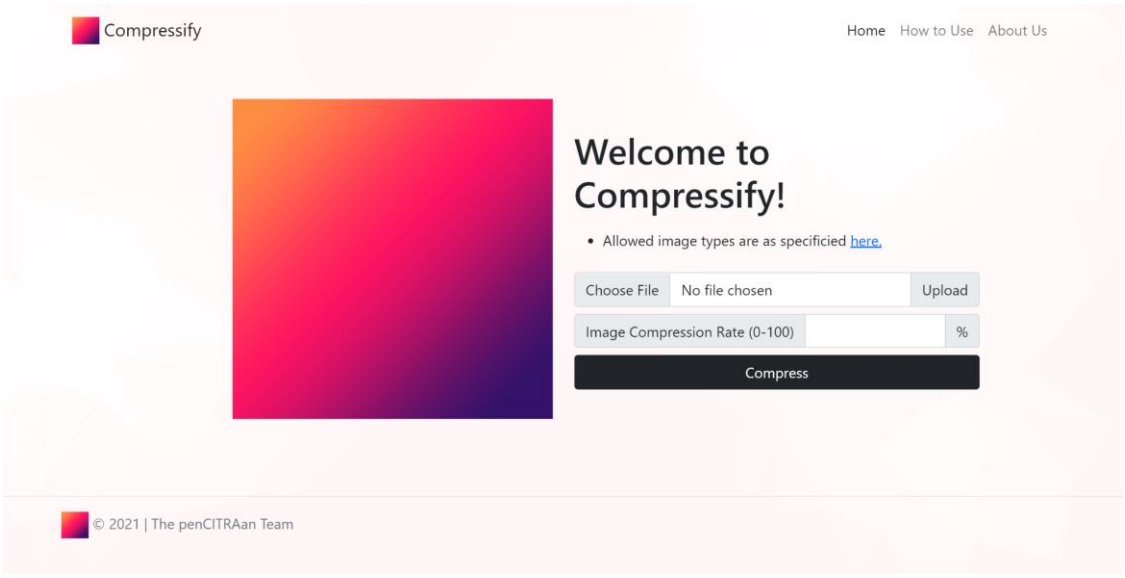
1. Compressify (home.html)

   Halaman ini merupakan halaman beranda dari aplikasi web yang digunakan oleh pengguna untuk mengkompresi gambar.

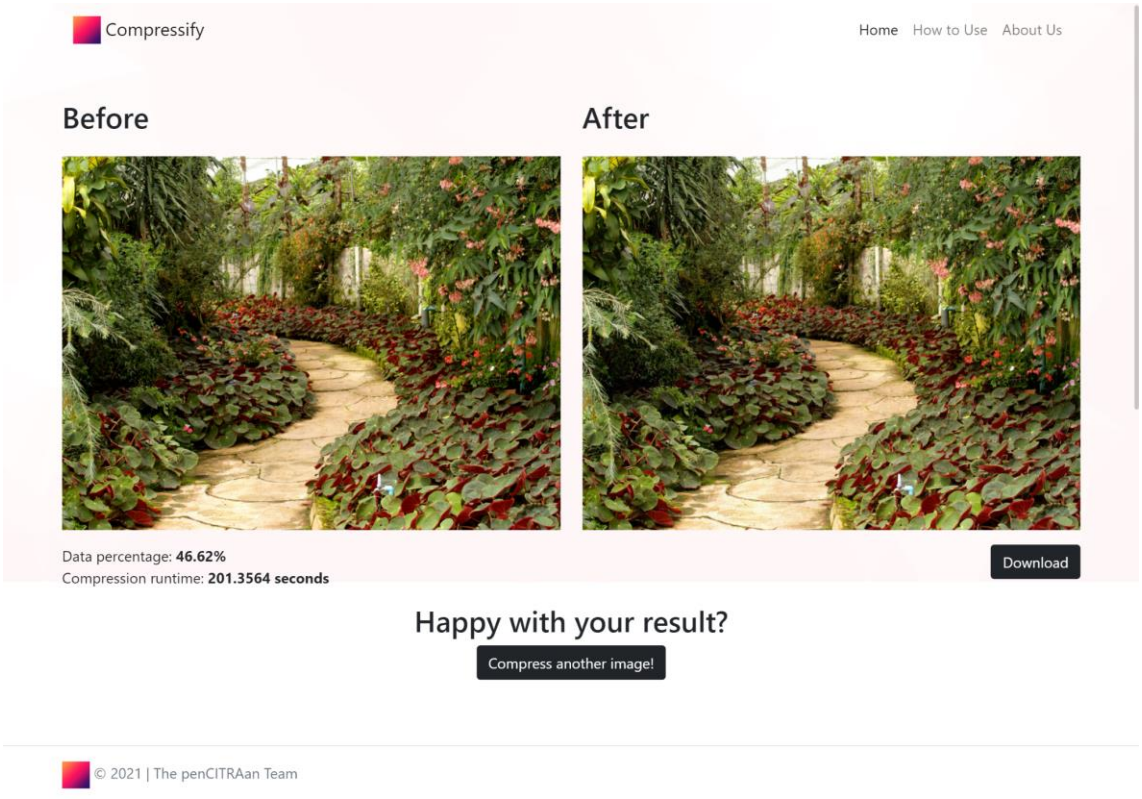


Terlihat di beranda ada form untuk mengunggah gambar, mengatur persentase kompresi, dan tombol ‘Compress’ untuk *submit* form untuk diterima oleh Flask. Bila user menekan tombol ‘Compress’ tetapi tidak mengunggah *file* atau membiarkan persentase kompresi kosong, maka akan keluar pop-up yang memerintahkan pengguna untuk mengisi bagian tersebut. “Please select a file” bila tidak ada *file* yang terpilih, dan “Please fill out this field” bila persentase kompresi kosong.

Bila pengguna mengunggah *file* dengan format yang tidak didukung (file format yang didukung ialah yang dapat diterima oleh library PILLOW untuk pemrosesan gambar) maka akan muncul halaman yang sama, namun dengan sebuah peringatan “*Allowed image types are as specified here.*” sebagai berikut.



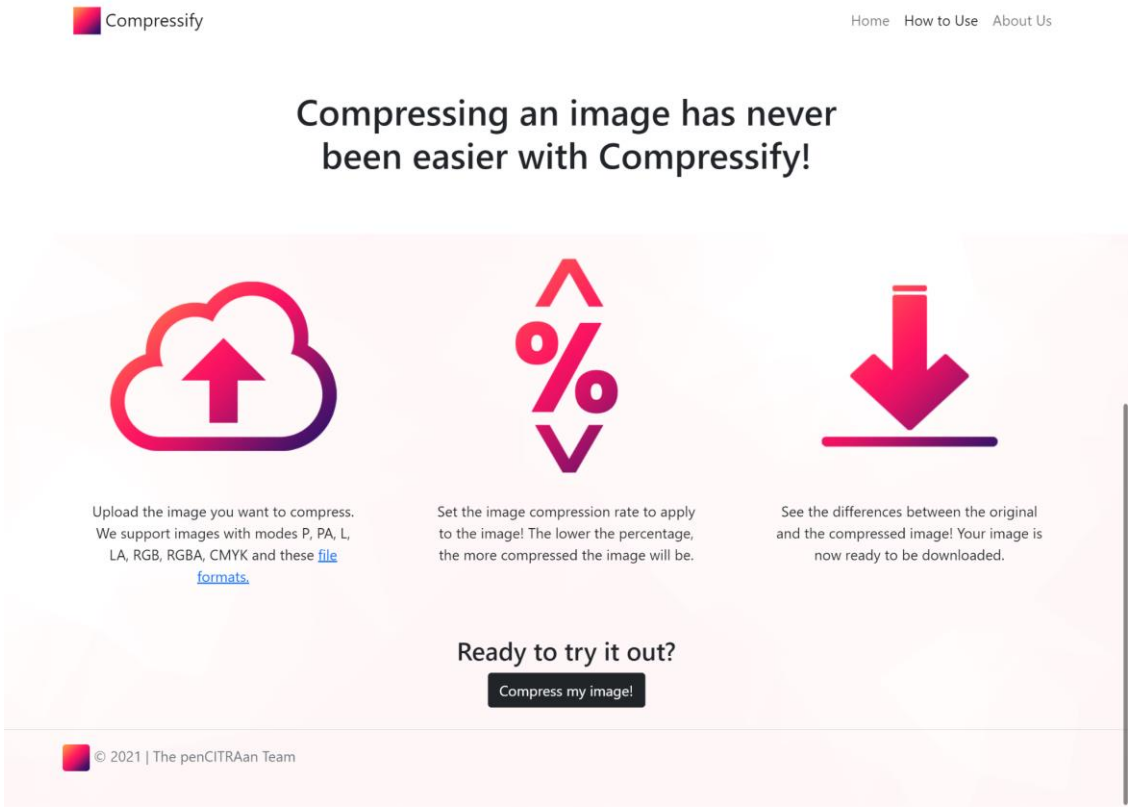
Bila pengguna mengunggah *file* dengan format yang benar dan mengatur persentase kompresi maka program kompresi SVD akan jalan dan ditampilkan hasilnya.



Ditampilkan gambar sebelum dan sesudah pemrosesan, persentase data dibandingkan originalnya, dan *runtime* program kompresi. Pengguna diberi pilihan untuk mengunduh gambar yang terkompresi menggunakan tombol “Download” dan mengkompres gambar lain menggunakan tombol “Compress another image!”.

2. How to use | Compressify (howtouse.html)

Halaman ini menunjukkan pengguna bagaimana cara memakai web ini yang dijelaskan dalam 3 langkah yang sederhana. Di bawahnya terdapat tombol yang mengalihkan pengguna ke halaman beranda untuk memulai kompres gambar.



3. About Us | Compressify (aboutus.html)

Halaman ini merupakan halaman “About Us” yang memberi pengenalan mengenai Compressify, sedikit cerita yang basa-basi dan seru dibalik pembuatan program, dan juga menjadi halaman apresiasi bagi tim dibalik proyek ini.

Compressify

Home How to Use About Us

Introducing, Compressify!

Gone are the days of lossless and big image files that hogs up your precious hard drive space!

— The penCITRAan Team *2021*

We have all gone around the internet searching for that good meme which tingles our meme hearts. Whether it be something you've seen in a recent PewDiePie video, accidentally stumbling upon it on an Instagram post, or making the effort to actually search for memes like a real memelord on the r/dankmemes subreddit, we all are a creature desperate for the need of memes. Whenever we see a meme we like, we express our gratitute for the meme by laughing ourselves out loud. It is then not uncommon for us to want to share this meme that made us laugh ourselves off, to our friends, our fellow memesters.

Problem is...

Sometimes that meme image that we wanted to share to our fellow memesters takes up a gazillion of space, having a file size somehow approaching close to infinity. This shocking revelation may and a lot of the time shatter our meme hearts and leave us feel disheartened.

This is where we come in!

We, the peNCITRAan Team, had noticed this growing problem over the past few years. More and more, image file sizes had grown at a multitude of levels beyond what it used to be, and that trend is not stopping any time soon. Something had to be done, and that something had to be done fast! If we do not solve this problem now, it will become harder to solve later on.

And that's what we did! We came up with an idea so original that it even blew our own minds off...

"What if we found a way to reduce the size of these images by means of compression?"

This simple question acted as a catalyst for us moving forward. Through countless hours of enduring pain and many more of endless exploration, we struggled our way back and forth in order to reach our final goal. A program to compress an image file. The holy grail itself. At the end, what we found to be THE WAY to compress an image was through the use of Singular Value Decomposition (SVD). Through the manipulation of matrices which represented every color and every pixel of an image, it turned out to be the sure-fire method for us to effectively compress an image file.

This project would not have been possible if it weren't for the amazing team of coders and thinkers behind it.

The penCITRAan Team

Gede Prasidha Bhawarnawa

13520004

Muhammad Helmi Hibatullah

13520014

Ahmad Alfani Handoyo

13520023

© 2021 | The penCITRAan Team

BAB IV
EKSPERIMEN

Pengujian Algoritma SVD dengan Symbolic Mathematics

Sebelum menggunakan algoritma QR untuk menemukan nilai dan vektor eigen, sebelumnya kami mencoba untuk menyusun suatu algoritma sehingga alur algoritma sesuai dengan metode dekomposisi SVD yang diajarkan di kelas IF2123 Aljabar Linear dan Geometri. Berikut adalah langkah-langkah alur kerja algoritma yang diharapkan:

1. Mengubah input gambar menjadi suatu matriks tiga-dimensi (panjang, lebar, *color-channel*).
2. Memisahkan setiap color-channel ke dalam suatu matriks tersendiri berdimensi dua. Setiap color-channel akan diproses secara seragam dan individual.
3. Mengambil matriks yang sudah dipisahkan dan mengalikannya secara *dot-product* dengan matriks itu sendiri yang sudah di-*transpose* sehingga terbentuk suatu matriks persegi. Matriks ini disimpan sebagai matriks AAT.
4. Membuat suatu matriks identitas dengan ukuran yang sama dengan matriks persegi pada langkah 3.
5. Mengalikan isi matriks identitas dengan suatu *symbolic expression* agar diagonal matriks bernilai dengan suatu variabel. Matriks ini disimpan sebagai matriks X.
6. Mengurangi matriks X dengan AAT (simpan sebagai matriks F) lalu mencari determinannya.
7. Menyamakan determinannya dengan nol dan mencari solusi polinomialnya. Solusi polinomial ini adalah nilai-nilai eigen singular kiri dari gambar.
8. Untuk setiap nilai eigen, ganti nilai variabel pada matriks F, lalu lakukan eliminasi Gauss-Jordan dengan matriks nol yang ukuran barisnya sama dengan matriks F dan hanya ada 1 kolom. Hasil dari eliminasi Gauss-Jordan ini adalah vektor eigen untuk nilai eigen yang bersangkutan.
9. Urutkan vektor eigen dan letakkan ke dalam suatu matriks baru sesuai nilai eigennya dari yang terbesar hingga yang terkecil. Matriks baru ini dinamakan matriks U.
10. Kalikan matriks awal dari langkah 3 yang sudah di-*transpose* dengan versi yang belum di-*transpose* dan simpan sebagai matriks ATA.

11. Membuat suatu matriks identitas dengan ukuran yang sama dengan matriks persegi pada langkah 3.
12. Mengalikan isi matriks identitas dengan suatu *symbolic expression* agar diagonal matriks bernilai dengan suatu variabel. Matriks ini disimpan sebagai matriks Y.
13. Mengurangi matriks Y dengan ATA (simpan sebagai matriks G) lalu mencari determinannya.
14. Menyamakan determinannya dengan nol dan mencari solusi polinomialnya. Solusi polinomial ini adalah nilai-nilai eigen singular kanan dari gambar.
15. Susun nilai eigen singular kanan yang tidak-nol dari paling besar ke paling kecil dan tempatkan pada diagonal suatu matriks identitas baru bernama matriks S.
16. Akarkan (pangkatkan dengan 0.5) setiap elemen pada matriks S.
17. Untuk setiap nilai eigen, ganti nilai variabel pada matriks G, lalu lakukan eliminasi Gauss-Jordan dengan matriks nol yang ukuran barisnya sama dengan matriks G dan hanya ada 1 kolom. Hasil dari eliminasi Gauss-Jordan ini adalah vektor eigen untuk nilai eigen yang bersangkutan.
18. Urutkan vektor eigen dan letakkan ke dalam suatu matriks baru sesuai nilai eigennya dari yang terbesar hingga yang terkecil. Matriks baru ini dinamakan matriks V.
19. Transpose matriks V dan simpan sebagai matriks VT.
20. Ulangi langkah 3 sampai 19 untuk setiap *color-channel* pada matriks gambar.
21. Setiap matriks yang didekomposisi dapat direkonstruksi ulang dengan mengalikan secara *dot-product* matriks U, S, dan VT sesuai urutan tersebut.

Berikut adalah implementasi algoritma untuk alur diatas:

```
48    def eigen_values(matrix):
49        """ Fungsi mengembalikan matriks berisi
50            nilai eigen.
51
52            Prekondisi: matrix berupa matriks persegi."""
53
54        x = symbols('x')
55        identity = create_identity(matrix)
56        identity_subtract_matrix = identity - matrix
57        eigen_val = sympy.Poly(identity_subtract_matrix.det()).all_coeffs()
58        eigen_val_np = np.array(eigen_val).astype(np.float64)
59        eigen_valuesssss = np.roots(eigen_val_np)
60        eigen_valuesssss = np.unique(eigen_valuesssss)
61        eigen_valuesssss = np.sort(eigen_valuesssss)[::-1]
62        eigen_val = Matrix([eigen_valuesssss])
-
63        return eigen_val
```

```
87    def singular_values(matrix):
88        """ Fungsi mengembalikan matriks berisi
89            nilai-nilai singular.
90
91            Prekondisi: matrix berupa matriks persegi."""
92
93        length, width = matrix.shape
94        final_singular_matrix = [[0 for _ in range(width)] for _ in range(length)]
95        A = matrix.T @ matrix
96        eigen_val = eigen_values(A)
97
-       for i in range(min(length, width)):
98  +     for i in range(eigen_val.cols):
99            if eigen_val[i] != 0:
100               final_singular_matrix[i][i] = eigen_val[i] ** 0.5
101
102       final_singular_matrix = Matrix(final_singular_matrix)
103
104       return final_singular_matrix
```

```
70      def create_identity(matrix):
71          """ Fungsi mengembalikan matriks identitas
72              yang tiap angka 1 diganti dengan simbol x.
73
74              Prekondisi: matrix berupa matriks persegi."""
75
76          length, width = matrix.shape
77          identity = Matrix.eye(length)
78          x = symbols('x')
79          for i in range(length):
80              for j in range(width):
81                  if (identity[i,j] == 1):
82                      identity[i,j] = x
83
84          return identity
```

```
10      def singular_vectors(matrix, left=True):
11          """ Menghitung matriks singular kiri atau kanan.\n
12              Untuk menghitung matriks singular kiri,
13                  Argumen matrix berupa matriks AAT dan left = True.\n
14              Untuk menghitung matriks singular kanan,
15                  Argumen matrix berupa matriks ATA dan left = False.\n
16
17              Prekondisi: matrix berupa matriks persegi."""
18          length, _ = matrix.shape
19
20          x = symbols('x')
21          identity = create_identity(matrix)
22          eigen_val = eigen_values(matrix)
23  +       print(eigen_val.cols)
24          zero_matrix = zeros(length,1)
25          final_eigen_vector = Matrix([[]])
26
27          for i in range(eigen_val.cols):
28              # ganti elemen diagonal dengan nilai eigen ke-i
29              for j in range(length):
30                  identity[j,j] = eigen_val[0,i]
31
32              # lakukan pengurangan kemudian dibulatkan (agar gauss jordan bekerja)
33              identity_subtract_matrix = np.array(identity - matrix).astype(np.float64)
34              identity_subtract_matrix = np.around(identity_subtract_matrix)
35              identity_subtract_matrix = Matrix(identity_subtract_matrix)
```

```
            # cari vektor eigen
+           tau0 = symbols(['tau0'])
            solutions, params = identity_subtract_matrix.gauss_jordan_solve(zero_matrix)
-           if (params.shape[0] != 0):
-               for p in params:
-                   # subtitusi 1 ke parameter pada solutions
-                   partial_solution_temp = solutions.xreplace({p:1})
-                   partial_solution = partial_solution_temp.xreplace({p:0 for p in params})
-                   # normalisasi vektor
-                   vector_len = 0
-                   for q in partial_solution:
-                       vector_len += q ** 2
-                   vector_len = (vector_len ** 0.5) ** (-1)
-                   partial_solution *= vector_len
-                   final_eigen_vector = final_eigen_vector.col_insert(length,partial_solution)
-           else:
-               final_eigen_vector = final_eigen_vector.col_insert(length, solutions)
-
+           print(f"solusi ke-{i}:", solutions)
+           print(f"param ke-{i}:",params)
+           for i in params:
+               print(type(i))
+
        if left:
            return final_eigen_vector
        else:
            return Matrix(transpose(final_eigen_vector))
```

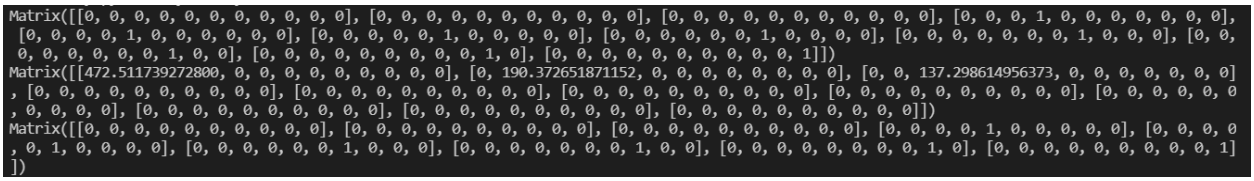
Untuk algoritma ini, dilakukan dua kali percobaan untuk pengujian. Percobaan pertama menggunakan matriks adalah matriks berukuran 2x3 dengan elemen [3,1,1] dan [-1,3,1]. Berikut adalah hasil yang diprintout oleh algoritma di atas:

```
Matrix([[0.707106781186547, -0.707106781186547], [0.707106781186547, 0.707106781186547]])
Matrix([[3.46410161513775, 0, 0], [0, 3.16227766016838, 0]])
Matrix([[0.408248290463863, 0.816496580927726, 0.408248290463863], [-0.894427190999916, 0.447213595499958,
 0], [-0.182574185835055, -0.365148371670111, 0.912870929175277]])
Matrix([[3.00000000000000, 1.00000000000000, 1.00000000000000], [-1.00000000000000, 3.00000000000000, 1.00
000000000000]])
```

Dengan urutan output adalah matriks U, S, VT, dan matriks input.

Jika dihitung secara manual, maka akan didapatkan matriks U,S,VT serupa sehingga dapat disimpulkan percobaan pertama berhasil.

Percobaan kedua menggunakan matriks 10x10 dengan tiga baris pertama berisikan elemen-elemen berikut: [49,48,198,230,0,0,0,0,0,0], [233,118,156,78,0,0,0,0,0,0], dan [0,118,255,63,0,0,0,0,0,0]. Sisa 7 baris berisikan nol semua. Berikut adalah hasil yang didapatkan.



Urutan matriks yang dihasilkan adalah U, S, dan VT.

Seperti yang terlihat, terdapat error dalam mengolah data singular kanan atau pun kiri untuk menghasilkan matriks U dan VT. Ini terjadi karena *library* sympy harus dikonfigurasi secara manual *floating point* yang dijadikan *pivot* untuk eliminasi Gauss-Jordan. Karena hasil perkalian matriks menghasilkan elemen-elemen dengan nilai tinggi, semua hasil dianggap oleh sympy sebagai 1.

Karena ketidakakuratan perhitungan dan karena tidak efektif untuk mengkonfigurasikan *floating point pivot* secara manual mengingat besarnya jumlah variasi hasil perkalian matriks, algoritma ini dinilai tidak memungkinkan untuk diimplementasikan.

Pengujian Algoritma QR dengan Jumlah Iterasi Berbeda

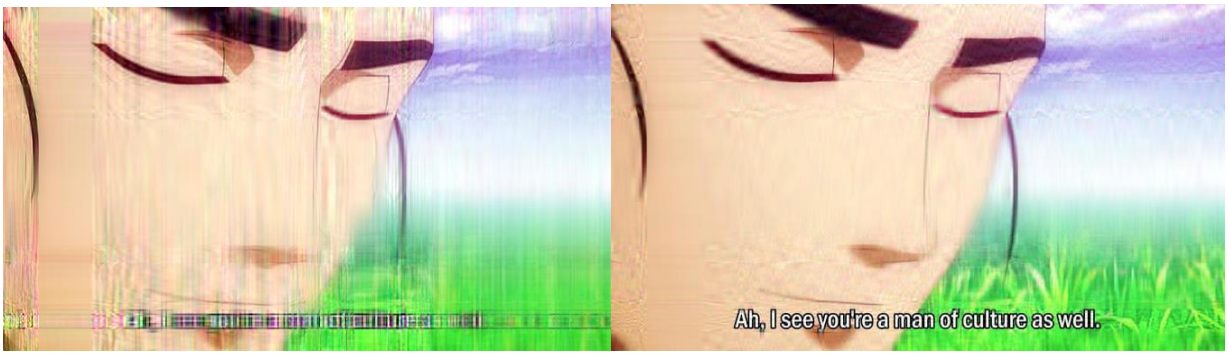
Pada algoritma yang berhasil kami susun dan telah kami jelaskan pada bab III, untuk mengaproksimasi nilai eigen dan vektor eigen menggunakan algoritma QR hanya dilakukan satu kali iterasi. Hal ini dilakukan dengan pertimbangan efisiensi waktu dan juga melihat hasil dari kompresi gambar yang tidak jauh berbeda. Kami telah mencoba mengubah-ubah jumlah iterasi pada proses mengkompres gambar mode RGB 680 x 383 dibawah ini dengan *compression rate* sebesar 5%.

Gambar yang Dijadikan Uji Coba.

Berikut grafik perbandingan jumlah iterasi dengan *runtime*.

Grafik Perbandingan Jumlah Iterasi dengan *Runtime.*

Dapat dilihat pada grafik di atas, bahwa untuk mengkompres gambar yang cenderung kecil dengan *compression rate* 5% pada jumlah iterasi 80 membutuhkan waktu lebih dari 20 detik. Sedangkan pada jumlah iterasi satu, maka waktu yang diperlukan hanya satu detik lebih sedikit.



Perbandingan Gambar pada Satu Kali Iterasi (kiri) dan delapan puluh Kali Iterasi (kanan).

Pada gambar yang hanya dilakukan satu kali iterasi, gambar tidak terlihat jelas, sedangkan pada delapan puluh kali iterasi gambar cukup terlihat jelas. Namun, untuk memperoleh kejelasan gambar seperti gambar yang di kanan ini diperlukan waktu dua puluh kali lipat lebih lama daripada satu kali iterasi. Menurut kami, hal ini tidak begitu baik karena untuk memperoleh tingkat kejernihan gambar yang sama pada satu kali iterasi, kita bisa meningkatkan *compression rate*-nya sekitar 5-10% dengan proses *runtime* yang cenderung sama.

Gambar yang Sudah Dikompres dengan *Compression Rate* 12% dan Satu Kali Iterasi. Untuk memperoleh gambar ini hanya diperlukan waktu sekitar 1,3599 detik.

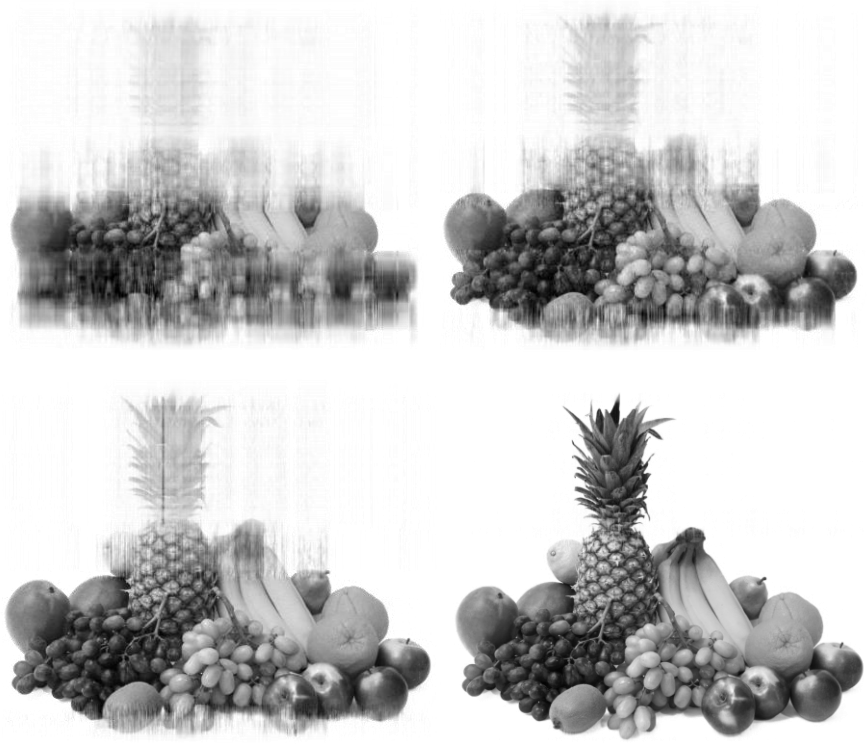
Pengujian Berbagai Mode Gambar

1. Mode Gambar L

Gambar dengan Mode L Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

Pada gambar diatas, terlihat bahwa semakin kecil *compression rate* yang digunakan, semakin rendah juga kualitas gambar yang dihasilkan. Gambar yang digunakan memiliki ukuran 974 x 1200 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-974. Pada *compression rate* 5%, nilai k adalah 0.05 x 974 yaitu 47 (dibulatkan kebawah). Dapat terlihat bahwa kualitas gambar pada k = 47 sangat rendah sampai wajah orang tidak terlihat jelas. Pada *compression rate* 20% atau k = 194, gambar terlihat jelas dengan sedikit penurunan kualitas. Sedangkan pada *compression rate* yang lebih tinggi dari itu, gambar tampak tidak ada penurunan kualitas dari gambar aslinya. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 9.5 detik.

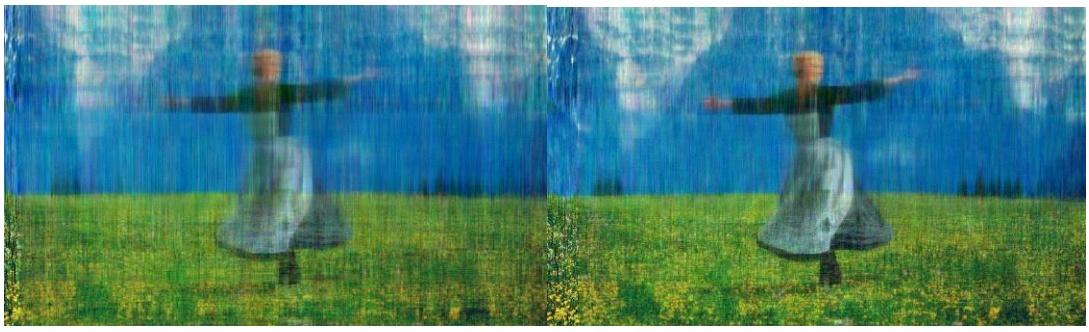
2. Mode Gambar LA



Gambar dengan Mode LA Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

Gambar yang digunakan memiliki ukuran 864 x 720 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-720. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 6 detik.

3. Mode Gambar P



Gambar dengan Mode P Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

Gambar yang digunakan memiliki ukuran 1189 x 700 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-700. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 1.3 detik.

4. Mode Gambar PA

Gambar dengan Mode PA Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

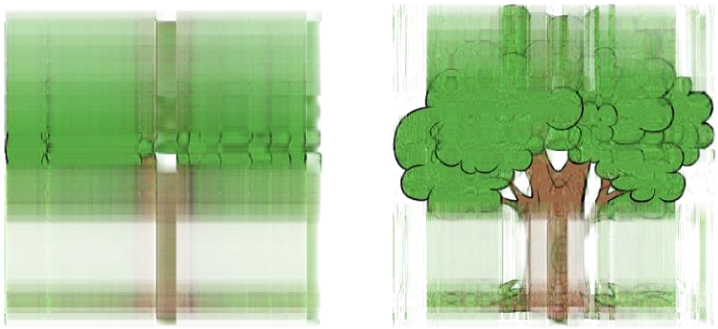
Gambar yang diujikan memiliki ukuran 2329 x 1706 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-1706. Gambar yang diuji dapat diperoleh dengan waktu runtime sekitar 226.4 detik atau sekitar 4 menit kurang.

5. Mode Gambar RGB

Gambar dengan Mode RGB Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

Gambar yang digunakan memiliki ukuran 680 x 383 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-383. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 1.3 detik.

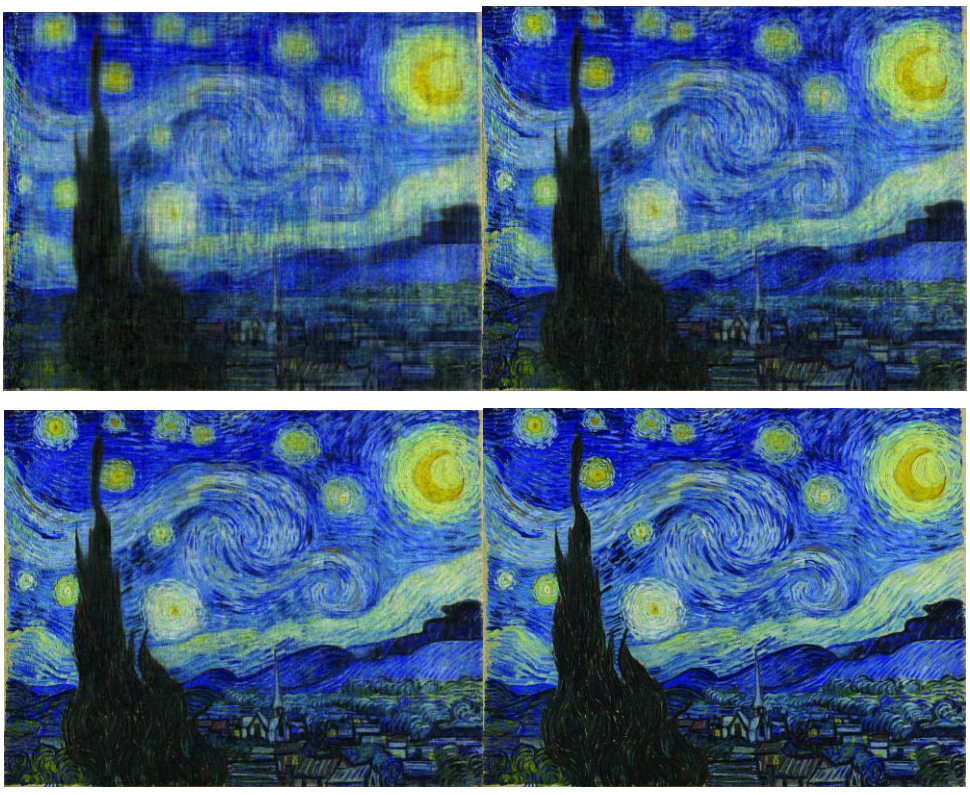
6. Mode Gambar RGBA



Gambar dengan Mode RGBA Dikompres dengan *Compression Rate* 11%, 20%, 30%, dan 40%.

Gambar yang digunakan memiliki ukuran 678 x 600 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-600. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 2.7 detik.

7. Mode Gambar CMYK



Gambar dengan Mode CMYK Dikompres dengan *Compression Rate* 5%, 10%, 20%, dan 40%.

Gambar yang digunakan memiliki ukuran 1280 x 1014 pixel sehingga nilai k yang dapat digunakan hanya dalam rentang 0-1014. Keempat gambar ini diperoleh dengan waktu *runtime* sekitar 19.2 detik.

BAB V
KESIMPULAN, SARAN, DAN REFLEKSI

Kesimpulan

Program yang telah dibuat adalah program kompresi gambar dengan memanfaatkan algoritma SVD dalam bentuk website lokal sederhana. Setelah dilakukan uji coba pada beberapa gambar, dapat disimpulkan bahwa program dapat berjalan sebagaimana fungsinya.

Saran

Setelah menyelesaikan pengerjaan tugas besar ini, beberapa saran yang dapat kami berikan adalah sebagai berikut. Disarankan menggunakan library pengolahan matriks yang tepat karena akan sangat berpengaruh dalam menentukan akurasi dari matriks SVD yang terbentuk. Seperti yang dibahas di dalam bab IV, ditemukan bahwa matriks dengan nilai elemen yang terlalu besar atau terlalu kecil tidak cocok jika diolah menggunakan *Python Library* sympy. Penggunaan sympy (khususnya dalam melakukan eliminasi Gauss-Jordan atau pembentukan matriks eselon baris tereduksi) hanyalah memungkinkan apabila *floating point pivot* dideklarasikan di dalam fungsi pengolahan secara manual. Selain itu, disarankan juga untuk membulatkan hasil atau elemen matriks yang diolah ke beberapa bilangan di belakang tanda koma. Tujuannya adalah agar data yang diproses tidak membutuhkan terlalu banyak memori dan data dapat diolah lebih cepat dan efektif dengan galat yang tidak terlalu besar.

Dalam menggunakan QR algorithm untuk menentukan nilai dan vektor eigen dari suatu matriks, perlu ada keseimbangan yang presisi antara banyaknya iterasi yang dilakukan dan juga akurasi dari nilai dan vektor eigen yang diinginkan. Tentu dengan dilakukannya iterasi yang semakin banyak, nilai dan vektor eigen yang didapat semakin mendekati kebenaran. Namun, *runtime* program akan jauh lebih lama.

Perlu dilakukan akal-akalan dan kompromi sehingga *runtime* algoritma menjadi lebih efisien. Di sini agar kami tidak menjalankan QR algorithm dua kali untuk menentukan U, S, dan VT, maka U dihitung berdasarkan matriks lainnya. Penentuan U dengan cara ini butuh mempersegikan matriks S (karena menjadi invers), sehingga SVD bukan lagi SVD murni melainkan menjadi salah satu jenis tereduksinya yaitu *Thin SVD*. Namun, metode ini

memungkinkan komputasi yang jauh lebih cepat dibandingkan SVD murni bila rank k kurang dari minimal panjang dan lebar dimensi gambar.

Refleksi

Pada proses pengerjaan tugas besar ini, tidak sedikit kendala yang kami hadapi. Bagi kami, mempelajari perkakas yang baru dikenal untuk membangun halaman web serta mengimplementasikan algoritma SVD yang baru diajarkan tidak lama setelah tugas ini rilis menjadi penghambat yang cukup besar. Kami sudah berulang kali gagal mengimplementasikan algoritma yang diajarkan di kelas sehingga kami harus merombak semua algoritma SVD hingga hari-hari akhir mendekati waktu pengumpulan.

Bagaimanapun juga, hambatan yang kami alami adalah suatu proses belajar sehingga dapat .diimplementasikan di dalam kehidupan pasca-kuliah nantinya.

REFERENSI

Munir, R. (2021). IF2123 Aljabar Geometri - Semester I Tahun 2021/2022.
http://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2021-2022/algeo21-22.htm

Hinno, R. (2021, February 22). QR decomposition.
https://ristohinno.medium.com/qr-decomposition-903e8c61eaab

Python. (n.d.). Python Imaging Library.
https://pillow.readthedocs.io/en/stable/handbook/concepts.html#concept-modes

Netlib. (n.d.). Decompositions.
http://www.netlib.org/utk/people/JackDongarra/etemplates/node43.html

ETH Zurich. (n.d.). The QR Algorithm.
https://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf

UCLA Edu. (2020). QR Algorithm.
http://www.seas.ucla.edu/~vandenbe/133B/lectures/qr.pdf

Quantstart. (n.d.). QR Decomposition with Python and NumPy.
https://www.quantstart.com/articles/QR-Decomposition-with-Python-and-NumPy/

Flask. (n.d.). API.
https://flask.palletsprojects.com/en/2.0.x/api/